1-3

УДК 004.8:519.254

В.В. Зубенко
Киевский национальный университет им. Тараса Шевченко, г. Киев, Украина

К понятию функции как вычислительной процедуре

Работа посвящена формальному уточнению общего понятия функции как вычислительной процедуры. При этом выбирается предельно возможный уровень абстракции, который условно можно было бы назвать пропозициональным: все объекты трактуются исключительно как теоретико-множественные «черные ящики». Понятие функции-процедуры позволяет уточнить и общее понятие абстрактного алгоритма – оно становится его производным. Рассматриваются общие свойства таких алгоритмов. Показано, что функции-процедуры и абстрактные алгоритмы замкнуты относительно регулярных композиций. Для табличных алгоритмов дано общее решение проблемы анализа.

Понятие функции – одно из самых важных понятий информатики. Известно два различных его толкования: процедурное (интенсиональное) и теоретико-множественное (экстенсиональное). В первом случае функция (от лат. functio – исполнение, совершение) рассматривается как способ для получения определенного результата на аргументах, во втором – как синоним теоретико-множественного отображения. Эти два подхода вошли в математическую практику не случайно. Они имеют принципиально разный смысловой оттенок. Если первый апеллирует к сути, содержанию конкретной функции, то второй – к тому, чем она отличается от других. Когда говорят о некотором отображении $f: A \to B$, то интересуются только тем, принадлежит ли пара (a,b) подмножеству $f \subset A \times B$, есть ли связь между элементами a и b или нет. При этом вопрос о природе этой связи, каким способом она осуществляется, считается несущественным. Фактически при экстенсиональном подходе осуществляется переход на более высокий уровень абстракции, при котором нивелируются индивидуальные черты конкретных объектов. В случае же функции-процедуры f на неё тоже смотрят как на возможность поставить в соответствие элементам множества A элементы множества B, но сам способ, реализация такого сопоставления выходит на передний план при определении f.

Исторически процедурный подход сформировался первым (сам термин предложил Г. Лейбниц (1646 – 1716)). Но он был почти вытесненный вторым в начале XX ст. в связи с тотальным переходом математики на теоретико-множественную основу. Достаточно сослаться на названия таких фундаментальных математических дисциплин, как функциональный анализ, теория функций комплексных переменных и т.д. Сегодня процедурный подход постепенно возвращает свои позиции благодаря бурному развитию конструктивной математики и информатики. Понятие функции как действия является центральным в теории алгоритмов, теории программирования, программологии и языках программирования.

Далее в тексте термин «функция» будет употребляться в обоих смыслах – и в традиционном теоретико-множественном, и как вычислительная процедура. Какой из них имеется ввиду – будет видно из контекста.

1 Функции как вычислительные процедуры

Пусть U – произвольное множество. Интуитивно функция f определяется как *процедура*, которая может быть применена к одному или нескольким элементам (аргументам) множества U с целью нахождения других элементов (результатов) с U. Применение функции состоит в проведении *вычисления* – последовательности определенных действий, которая завершается построением результата. О том, как происходит выбор этой последовательности, чем он определяется, речь пойдет ниже. А пока заметим, что функция не всегда гарантирует однозначность, и даже существование результата. Это означает, что процесс вычисления может не привести к результату, а применение функции повторно к тем же аргументам может приводить к другому результату. В этом смысле все функции разделяются на *детерминированные* и *недетерминированные*. Первые гарантируют однозначность своих результатов, все остальные принадлежат ко вторым. Как видим, 1) вычислительное пространство, в котором развертываются вычисления, является как минимум двумерным – с временной и информационной координатами и 2) существует механизм для определения результатов вычислений.

Термы $f(x)$, fx и xf используются для записи результата применения функции f к аргументу x. Учитывая, что результата такого применения по определению может и не быть, будем добавлять к указанным термам справа символ $\downarrow$ в случае, если данный результат гарантированно существует. С каждой функцией f связаны две области – *определения* $D_f = \{x \in U : f(x)\downarrow\}$ и *значений* $E_f = \{y \in U : y = f(x) \,\&\, x \in D_f\}$. Чтобы показать эту связь, функцию f подают в виде $f : D_f \to E_f$ и называют *всюду определенной* на D_f. На практике бывает проще описать не сами области определения и значений данной функции, а определенные их надмножества, например, T и R. Чтобы подчеркнуть, что функция рассматривается над этими более общими областями, ее записывают как $f : T \to R$ и называют *частичной*, а пару (T, R) – *типом* функции. Будем считать выражение $f(x)$ бессодержательным (т.е. равным $\#$) для всех аргументов вне области определенности функции. Функции $f : A \to B$ и $g : C \to D$ называются *эквивалентными* $(f \cong g)$, если $\forall x \in A \cup C \quad fx = gx$ (*). Например, функции $f_0 : N \to N$ и $f : N \to N$, определяемые термами $f_0 x \overset{def}{=} x / x$ и $fx \overset{def}{=} 1$, не эквивалентны. Их разделяет точка $x = 0$. Эквивалентные функции называются *равными*, если их типы совпадают.

Вернемся к интуитивному определению функции и попытаемся его детализировать. Как следует из этого определения, для уточнения функции необходимо (и достаточно!) уточнить понятия процедуры и вычисления.

Если взять одно отдельное вычисление, то оно не является одноразовым актом, а наоборот, начавшись в определенный фиксированный момент времени с определенной начальной информацией на входе, развертывается во временном пространстве и сопровождается выполнением определенных элементарных действий и определением результата. Пусть $<\Gamma, <>$ – произвольная линейно упорядоченная совокупность, которую будем трактовать как временное пространство[1]. Обозначим τ^+ и τ^- – непосредственно следующий и предыдущий моменты времени относительно момен-

[1] В некоторых случаях часовое пространство может иметь более сложную структуру [1].

та τ. Считается, что во временном пространстве нет наименьшего и наибольшего элементов. Информационную составляющую процессов вычислений представляют так называемые *состояния*. На данном уровне абстракций нас не интересует их структура. Пусть S – произвольное множество состояний.

Двойка $\Pi = \langle \Gamma, S \rangle$ называется *вычислительным пространством*, а пара $(\tau, s) \in \Gamma \times S$ – *конфигурациями* пространства. Пусть $\tau, \tau_1, \tau_2 \in \Gamma$. Положим $\Gamma_\tau = \{\tau' \in \Gamma \mid \tau \le \tau'\}$, $\Gamma_\tau^+ = \Gamma_\tau \setminus \{\tau\}$, $\Gamma_\tau^- = \Gamma \setminus \Gamma_\tau$, $[\tau_1, \tau_2] = \{\tau : \tau_1 \le \tau \le \tau_2\}$. *Абстрактным процессом* в вычислительном пространстве с началом в момент времени τ называется произвольное отображение вида $p : \Gamma \to S$, а *абстрактным процессом на интервале* $[\tau_1, \tau_2]$ – произвольное отображение вида $p : [\tau_1, \tau_2] \to S$. Состояния $p\tau$, $p\varsigma$ ($\tau_1 < \varsigma < \tau_2$) и $p\tau_2$ называются *начальным, промежуточными* и *заключительным* состояниями процесса p. Возьмем произвольную функцию $f : A \to B$ и ее аргумент a. Если процесс p на интервале $[\tau_1, \tau_2]$ начинается в состоянии a, а заканчивается в заключительном состоянии $b = fa$, говорится, что он *представляет* функцию f на аргументе a. Говорят, что совокупность процессов на интервалах *представляет* функцию f, если для каждого ее аргумента из области определения эта совокупность содержит процесс, который ее представляет на данном аргументе.

Теперь среди всех абстрактных процессов в вычислительном пространстве Π попробуем выделить процессы-вычисления и их совокупности. Индексом многозначной функции назовем максимальную мощность множеств значений функции на отдельно взятых аргументах. Зафиксируем некую совокупность $\Delta = \{f_i : S \to S \mid i \ge 0\}$ *элементарных операций* на состояниях и определим *функцию управления* процессами. Последняя с каждым моментом времени процесса связывает одно или несколько возможных элементарных преобразований его состояния. Положим ее как имеющую вид $\delta : \Gamma \times S \to 2^\Delta$ и конечный индекс[1].

Вычислительной процедурой над пространством Π с совокупностью элементарных преобразований Δ и функцией управления δ называется тройка $P = \langle \Pi, \Delta, д \rangle$.

Каждая вычислительная процедура порождает определенную совокупность процессов вычисления в пространстве Π. Пусть $\tau \in \Gamma$ и $s \in S$ – произвольные моменты времени и состояние. **Вычисления** $p : \Gamma_\tau \to S$ в процедуре P с началом в момент времени τ и начальным состоянием s_0 определяются рекуррентно: $p\tau = s_0$ и для всех $\varsigma > \tau$ $p\varsigma = f_\varsigma(p\varsigma^-)$, где $f_\varsigma \in \delta(\varsigma, p\varsigma^-)$. Функция управления по текущему моменту времени ς и состоянию $p\varsigma^-$ в предыдущий момент времени определяет совокупность элементарных операций, которые могут быть применены для получения текущего состояния процесса. Будем говорить, что вычисление p в момент времени ς *переходит* от предыдущего состояния $p\varsigma^-$ к следующему состоянию $p\varsigma$, и обозначать этот факт $p\varsigma^- \to_P p\varsigma$. Вариантов таких переходов может быть несколько, поскольку функция перехода неоднозначна. В действительности их может быть: а) два и более (в случае *недетерминированных* процедур); б) ровно один (в случае *детерминированных* процедур); в) ни одного (вычисления останавливаются).

[1] В общем случае функция управления может задавать преобразование и часового компонента (при этом возможны «прыжки» процесса во времени как вперед, так и назад [1]).

Вычисления p по процедуре P на интервале $[\tau_1, \tau_2]$ определяются как ограничение соответствующего вычисления p по процедуре P с началом в момент τ_1 на временной интервал. $[\tau_1, \tau_2]$ Будем говорить, что вычисление p в этом случае начинается состоянием $p\tau_1$ и заканчивается состоянием $p\tau_2$,и обозначать этот факт $p\tau_1 \Rightarrow_P p\tau_2$.

Обычно интерес представляют не все возможные вычисления в процедуре, а прежде всего те, что начинаются в определенный фиксированный момент времени с определенных выделенных входных состояний и заканчиваются определенными выделенными заключительными состояниями. Поэтому введем понятие *инициальной процедуры* $P(S_0, S_{fin}, \tau_0)$ над пространством П как четверки $\langle P, S_0, S_{fin}, \tau_0 \rangle$, где $P-$ определенная процедура над пространством Π, $S_0 \subseteq S$ и $S_{fin} \subseteq S$ – выделенные подмножества соответственно *входных* и *заключительных* состояний, $\tau_0 \in \Gamma$ – начальный момент времени. *Результативными* назовем вычисления в процедуре P, которое начинается в момент времени τ_0 с определенного начального состояния $p\tau_0 = s_0 \in S_0$, заканчивается в определенный момент τ заключительным состоянием $p\tau \in S_{fin}$, и все промежуточные состояния которого незаключительные. Состояние $p\tau$ называется *результатом* данного вычисления на s_0, обозначается $P^*(s_0)$ и есть единственным, если функция управления – однозначная. Если в результативном вычислении $p\tau_0 \Rightarrow_P p\tau$ снять условие, чтобы все промежуточные состояния были незаключительными, то о таком вычислении будем говорить как о гиперрезультативном. Его исходные состояния могут встречаться и среди промежуточных, а заключительный обозначается $P^{**}(s_0)$. В некоторых случаях как результат вычисления берется не само заключительное состояние, а значение на нём некоторой *результирующей* операции $Val : S_{fin} \to B$, а также считают заключительными и те конечные вычисления, на последних состояниях которых не определена функция управления. Если значение результирующей операции Val на заключительном состоянии не определено, то оно считается *безрезультатным*. Таковыми являются, як правило, и бесконечные вычисления, а также те конечные, которые не имеют результативных продолжений.

Мы уточнили понятие процедуры и вычисления в ней. Это позволяет уточнить и понятие функции как процедуры.

Функциями-процедурами называются инициальные вычислительные процедуры. С каждой функцией-процедурой $P(S_0, S_{fin}, \tau_0)$ связаны два ее *графика* – соответствия на входных состояниях $P^* : S_0 \to S_{fin}$ и $P^{**} : S_0 \to S_{fin}$, а именно те, значения которых на начальном состоянии $s_0 \in S_0$ составляют заключительные состояния соответственно всех результативных и гиперрезультативных вычислений с началом в момент τ_0. О графиках P^* и P^{**} будем говорить как о таких, которые *представляет* или *вычисляет* функция P.

Много примеров функций-процедур можно найти в математике, в частности, в алгебре и геометрии. Например, задачи по планиметрии на построение с помощью циркуля и линейки являются фактически задачами на построение определенных (не обязательно детерминированных) функций. Состояниями здесь есть совокупности

фигур на плоскости, которые могут быть построены с помощью циркуля и линейки и операций выделения произвольной точки плоскости, точек пересечения фигур, операций именования точек и фигур, а также условных вариантов подобных операций. Временное пространство дискретное и совпадает с натуральными числами. Интересная ситуация возникает в случае существования нескольких решений задачи. Тогда общее решение может обеспечить недетерминированная функция. Но ее можно и детерминизировать. Если, например, для решения задачи используется школьная доска и необходимо построить два разных решения, то после завершения построения первого из них доска вытирается (полностью или частично) и функция возвращается в одно из предыдущих состояний доски, но, что важно, не в предыдущую конфигурацию – момент времени другой! Благодаря этому процесс вычисления может быть продолжен детерминированно, как того требует построение второго решения.

Функции-процедуры, управление действиями в которых реально зависит от временной компоненты, будем называть *темпоральными* в отличие от *автоматных*, когда это не так.

Классическими примерами автоматных процедур являются конечные и магазинные автоматы, машины Тьюринга, игровые исчисления и т.д. Если взять последние, то иногда в их правилах и стратегиях временные факторы могут играть важную роль и они приобретают признаки темпоральных процедур. Во многих играх (шахматы, шашки, карты), например, важным фактором есть очередность хода. В шахматах (шашках) во все нечетные моменты времени ходят белые, а в четные – черные. Имеются и другие обстоятельства, связанные со временем. Например, рокировка короля в шахматах зависит от того, двигался ли он до этого момента игры. В некоторых моделях (игра с часами), на стратегию выбора очередного хода влияет ограниченность времени на часах (игра в цейтноте). В последнем случае на временном пространстве игровой функции определяют некоторую топологию.

Обратимся к натуральной арифметике и рассмотрим функцию (алгоритм Евклида) GCD для нахождения наибольшего общего делителя НСД(a,b) двух натуральных чисел.

Пусть $\Pi = \langle \Gamma, S \rangle$ – вычислительное пространство с натуральным временем $\Gamma = N$, множеством состояний $S = N^+ \times N^+$, $N^+ = N \setminus \{0\}$. Набор операций $\Delta = \{\alpha, \beta\}$ и функцию управления δ определим так. Для произвольных $a, b \in N^+$, $\tau \geq 0$ положим: $\alpha(a,b) = (a-b,b)$, $\beta(a,b) = (a,b-a)$, $\theta(a,b) = (a,b)$, $\delta(\tau,(a,b)) = (a \neq b \to (a > b \to \alpha \mid \beta) \mid \theta)$, где $(p \to a \mid b)$ равно a, если $p \neq 0$ и b для $p = 0$.

Определим функцию $GCD = \langle \Pi, \Delta, \delta \rangle$ и ее инициализированный вариант $GCD\,(S_0, S_{fin}, 0)$ с результирующей операцией проектирования по первой компоненте pr_1, где $S_0 = N^+ \times N^+$, $S_{fin} = i_N$. Несложно проверить, что полное вычисление функции GCD на каждом входном состоянии $s_0 = (a,b)$ будет результативным. Действительно, непосредственно из определения GCD и соотношений:

1) $\gcd(d,d) = d$,

2) $(b > a \Rightarrow \gcd(a,b) = \gcd(a,b-a))$, 3) $(b < a \Rightarrow \gcd(a,b) = \gcd(b,a-b))$

следует, что последнее состояние в вычислении функции GCD на входном состоянии $s_0 = (a,b)$ будет иметь вид (d,d) для некоторого числа $d = \gcd(a,b)$, потому что рано или поздно компоненты состояния процесса станут равными, а операции α и β не изменяют наибольший общий делитель компонентов состояния. Конечный

результат вычисления формирует операция проектирования $\mathrm{pr}_1(d,d)=d$. Таким образом, $GCD^*(a,b)=\mathrm{gcd}(a,b)$.

Отдельного рассмотрения заслуживают инициальные процедуры, которые заканчивают вычисления исключительно по временным критериям. В этом случае могут даже не фиксироваться заключительные состояния процедуры, а вместо них на состояниях, например, вводится определенный частичный порядок и как результат вычисления (в том числе и бесконечного) рассматривается наименьшая верхняя грань его состояний. Другой вариант составляют процедуры, которые заканчивают работу в определенный момент времени τ или после некоторого момента времени τ и т.д. Функции P и Q называются *эквивалентными* ($P\equiv Q$), если их графики P^* и Q^* эквивалентные. Состояние функции $P(S_0,S_{fin},\tau_0)$ назовем *допустимым*, если оно встречается среди состояний некоторого результативного ее вычисления. Положим $S_{acc}\subseteq S$ – подмножество всех допустимых состояний функции $P(S_0,S_{fin},\tau_0)$. Для соответствий, вычисляемых функцией, состояния за пределами подмножества S_{acc} несущественны. Пусть $P'(S'_0,S'_{fin},\tau_0)$ – функция над пространством Π' с множеством состояний S_{acc} и ограниченными на Π' элементарными операциями, функцией перехода, входными и заключительными состояниями функции P. Очевидно, что процедуры P и P' эквивалентные. Функция, все состояния которой допустимы, называется *приведенной*. Например, ранее определенная функция GCD является приведенной.

При конструировании функций важно иметь специальные операции, которые из одних функций строят другие, более сложные. Такие операции называются *композициями.* Наиболее известны три из них – последовательное выполнение, разветвление и итерация. Они называются *регулярными* и входят в состав композиций систем алгоритмических алгебр. Имеет место

Теорема 1 (замкнутости). Класс функций замкнут относительно регулярных композиций.

Доказательство теоремы приведено [1]

2 Абстрактные алгоритмы

При определении понятия алгоритма мы хотим отойти от принятой в теории алгоритмов практики определять какой-то конкретный класс алгоритмов (например, машины Тьюринга или алгоритмы Маркова) и, ссылаясь на его универсальность, связывать с ним общее понятие алгоритма. Такой подход, имея определенный смысл в мировоззренческом отношении, к сожалению, почти ничего не дает непосредственно программированию. Поэтому мы предлагаем другой подход, опирающийся на понятие функции и её реализацию.

Когда говорят о реализации класса функций или отдельной функции, то имеют в виду, что существует исполнитель (реальный или гипотетический), который способен воспринимать функции и производить в них конкретные вычисления: фиксировать отдельные состояния, находить значения функции управления на каждом шаге вычисления и выполнять соответствующие элементарные преобразования. Для представления процедур и их компонентов исполнитель имеет специальный внутренний язык программирования (далее – просто *внутренний язык*). Он не обязательно вербальный, но обязательно *финитный* – считается, что исполнитель способен воспринимать и оперировать только конечными с точки зрения описания объектами.

Объекты внутренних языков исполнителей, которые реализуют функции, будем называть **конструктивными,** представленные в них функции – **абстрактными алгоритмами** (**А-алгоритмами**), а сами внутренние языки – **алгоритмическими языками**[1]**.**

Таким образом, конструктивность объекта имеет смысл только в контексте некоторого исполнителя и его алгоритмического языка, а функция-процедура становится А-алгоритмом только при условии, что для неё существует конкретный исполнитель. На выбранном нами пропозициональном уровне абстракции не рассматривается внутренняя структура самих исполнителей. Мы только знаем, что исполнитель или воспринимает конструктивный объект как входящий, или строит его с помощью конечного числа элементарных преобразований в процессе вычислений.

Как пример класса А-алгоритмов можно привести обычные арифметические выражения и общие интерпретированные Ω-термы. Каждый из них определяет некоторую функцию-процедуру для вычисления значений. Исполнителем таких А-алгоритмов может быть, например, лицо, умеющее анализировать структуру термов и выражений и выполнять соответствующие операции и предикаты. Можно пойти еще дальше и рассмотреть все интерпретированные регулярные выражения над сигнатурой Ω.

Соответствия A^* и A^{**}, вычисляемые А-алгоритмом (его графики), называются *алгоритмически вычисляемыми*. Как и процедуры, А-алгоритмы могут быть детерминированными и недетерминированными. Приведем основные свойства А-алгоритмов, которые непосредственно вытекают из их определения.

Массовость. А-алгоритм может быть применен к любому входному состоянию.

Темпоральность. Вычисления А-алгоритма происходят во временном пространстве, элементы которого могут влиять на выбор преобразований текущего состояния.

Элементарность. В каждый момент времени в вычислении выполняется одна элементарная операция из фиксированной совокупности таких операций.

Определенность. Порядок применения элементарных операций в вычислении не является произвольным, а определяется функцией управления.

Результативность. Есть механизм завершения вычислений.

Финитность. Конечность представления А-алгоритма.

Релятивность. Относительность понятия А-алгоритма. Конструктивность А-алгоритма как объекта алгоритмического языка напрямую зависит от конкретного исполнителя. А-алгоритм для данного исполнителя не обязательно будет А-алгоритмом для другого. Другим аспектом релятивности А-алгоритмов являются взаимоотношения элементарных операций и исполнителя. В некоторых случаях элементарные операции (все или их часть) могут считаться абстрактными и не подлежать исполнению. Такие операции называются *оракулами*, а сами А-алгоритмы – алгоритмами с *оракулами*.

Как видим, А-алгоритмам присущи основные свойства классических числовых и словарных алгоритмов [2]. Однако появились и новые специфические черты – темпоральность и релятивность.

В связи с релятивностью возникает задача сравнения различных классов алгоритмов по мощности. Будем говорить, что алгоритм A_1 с множеством состояний S_1 является *моделью* алгоритма A_2 с множеством состояний S_2 относительно функции кодирования $\varphi : S_2 \to S_1$, если для каждого из входных состояний s алгоритма A_2 выполняется: $A_2^*(s) = \varphi^{-1}(A_1^*(\varphi s))$. Пусть K_1 и K_2 – произвольные классы алгоритмов

[1] Не случайно первые языки программирования тоже назывались алгоритмическими. Например, Алгол-60 (англ. – ALGOrithmic Language ALGOL-60).

над вычислительными пространствами $\Pi_1 = \langle \Gamma_1, S_1 \rangle$ и $\Pi_2 = \langle \Gamma_2, S_2 \rangle$ соответственно. Зафиксируем определенную функцию кодирования $\varphi : S_2 \to S_1$. Говорят, что класс алгоритмов K_1 *моделирует* класс алгоритмов K_2, если для каждого алгоритма с K_2 в классе K_1 есть его модель. Если классы алгоритмов моделируют друг друга относительно определенных фиксированных функций кодирования, то они называются *эквивалентными* относительно этих функций.

Фиксируя конкретные вычислительные пространства Π и соответствующих исполнителей, можно получить весь спектр классических алгоритмических систем (машины Тьюринга, алгоритмы Маркова и Колмогорова – Успенского, дискретные преобразователи, различные классы автоматов и схем программ, формальные грамматики и исчисления и т.п.) [2]. Все перечисленные классы классических алгоритмических систем являются, как известно, эквивалентными и получили название *универсальных*.

Посмотрим, например, на машины Тьюринга как на А-алгоритмы. Исполнитель А-алгоритма A, реализующего машину Тьюринга, оперирует состояниями, изображающими машинные конфигурации. Такие состояния имеют две компоненты: первый компонент – внутреннее состояние машины, второй – подает состояние памяти машины и имеет вид последовательности ячеек, в которой хранится слово в алфавите X с указанной позицией рабочей ячейки. Временное пространство Γ – натуральное. Значение функции перехода на конфигурации определяется внутренним состоянием и состоянием рабочей ячейки (рабочим состоянием). Оно никоим образом не зависит от номера шага (временной компоненты) вычисления. Само действие состоит в изменении исполнителем внутреннего и рабочего состояний и позиции рабочей ячейки. Новой позицией может стать позиция одной из двух непосредственно соседних ячеек в последовательности. Учитывая, что совокупности внутренних состояний Q и состояний рабочей ячейки X машин Тьюринга конечны, то функцию перехода нашей процедуры можно описать тоже конечно, например, таблично как соответствие $\delta : Q \times X \to Q \times X \times \{-1, 0, 1\}$, которое по текущим внутренним и рабочим состояниями определяет новые их значения и новую позицию рабочей ячейки. Ноль означает, что она не меняется, –1 означает сдвиг влево, а 1 – сдвиг вправо. Выделяется начальный $q_0 \in Q$ и совокупность заключительных $F \subseteq Q$ внутренних состояний. Совокупности S_0 начальных и S_{fin} заключительных состояний А-алгоритма составляют множества $\{q_0\} \times X^*$ и $F \times X^*$ соответственно. Результатом вычисления процедуры P на начальном состоянии есть состояние ее рабочей ленты в заключительном состоянии вычисления. Учитывая, что начальное внутреннее состояние А-алгоритма фиксировано, то можно полагать, что он вычисляет словарную функцию A^* на множестве рабочих состояний X^*.

Пусть $X = \{a, \#\}$ и $F = \{q^*\}$. Табл. 1 задает А-алгоритм для машины Тьюринга, которая вычисляет функцию $f(a^n \# a^m) = a^{n+m}$, где n, m – произвольные натуральные числа:

Q	X	$Q \times X \times \{-1, 0, 1\}$
q_0	a	$(q_1, \varepsilon, 1)$
q_0	#	$(q^*, \varepsilon, 0)$
q_1	a	$(q_1, a, 1)$
q_1	#	$(q^*, a, 0)$

Стоит отметить, что машины Тьюринга, как и другие классические алгоритмы, являются сугубо *автоматными* – выбор следующего действия в них не зависит от текущего момента времени.

Для А-алгоритмов тоже имеет место

Теорема 2 (замкнутости). Класс А-алгоритмов замкнут относительно регулярных композиций.

Доказательство. Следует из доказательства Теоремы 1.

Как и классические алгоритмы, А-алгоритмы могут использоваться для описания множеств. Подмножество состояний $R \subseteq S$ называется *разрешимым* (*частично разрешимым*), если существует А-алгоритм A, график которого совпадает с характеристической функцией χ_R (частичной характеристической функцией $\overline{\chi}_R$) подмножества R.

А-алгоритмы вычисляют и временные операторы, которые отображают их входные конфигурации в заключительные, поэтому они могут служить основой для уточнения и общего понятия вычислимого временного оператора в произвольной области. Этот путь прямой и не опирается на те или иные универсальные числовые или словарные модели алгоритмов.

3 Табличные алгоритмы

Важный класс А-алгоритмов составляют табличные А-алгоритмы. Пусть временное пространство Γ изоморфно аддитивной группе Z целых чисел и пусть A – произвольная процедура над пространством $\Pi = \langle Z, S \rangle$. Факторизуем функцию управления по обоим аргументам. Зафиксируем некоторую часовую константу $k \geq 0$ и некоторое отношение эквивалентности π на состояниях с S. Будем говорить, что функция управления δ периодичная по модулю отношения π с периодом k (или просто k-периодичная по модулю π), если для всех эквивалентных состояний p и s и произвольного момента времени $t \in Z$: $\delta(t, p) = \delta(t \pm k, s)$. Функция управления δ – просто периодичная по модулю π, если она k-периодичная по модулю π для некоторого k. Функция-процедура с периодичной функцией управления, а эквивалентность π – конечный индекс $|\pi|$ (конечное число классов эквивалентности) и разрешимые классы, называется *табличным алгоритмом.* Табличные алгоритмы (T – алгоритмы) можно задавать двумерными таблицами размером $k \times |\pi|$: первое измерение отвечает числам от 0 до $k-1$, второе – классам эквивалентности π, а в клетках таблицы размещены значения функции управления. Традиционные алгоритмические системы, как правило, являются табличными как А-алгоритмы. А поскольку они являются автоматными, то и – 1 – периодичными по модулю равенства. Это видно на примере конечных и МП-автоматов. Машина Тьюринга – табличный алгоритм: два её состояния эквивалентны, если их внутренние состояния и состояния рабочих ячеек на лентах совпадают.

Для табличных алгоритмов тоже имеет место

Теорема 3 (замкнутости). Класс табличных алгоритмов замкнут относительно всех регулярных композиций.

Доказательство приведено в [2].

Проблема *анализа* А-алгоритмов, которые принадлежат некоторому классу K, состоит в поиске тех или иных общих алгебраических форм представления для их графиков. Следующая теорема решает эту проблему для табличных алгоритмов. Кон-

фигурации называются *соседними*, если они принадлежат одному временному периоду (их временные компоненты t_1 та t_2 удовлетворяют условию: $t_1 \div k = t_2 \div k$, где $\div$ – операция целочисленного деления).

Теорема 4 (о регуляризации графиков табличных алгоритмов). Пусть A – табличный алгоритм над некоторым вычислительным пространством $\Pi = \langle Z, S \rangle$. Тогда его график регулярен относительно базиса, который состоит из элементарных операций Δ, характеристических предикатов классов разбиения отношения π, множеств входных и заключительных состояний, а также предиката временного соседства конфигураций.

Доказательство приведено в [3].

Заключение

В работе предложено формальное уточнение общего понятия функции как вычислительной процедуры. Это позволяет уточнить и общее понятие абстрактного алгоритма – оно становится его производным (сужением). Рассмотрен важный класс А-алгоритмов – табличные алгоритмы. Все основные классические алгоритмические и автоматные системы являются табличными алгоритмами. Показано, что функции и абстрактные алгоритмы замкнуты относительно регулярных композиций. Для табличных алгоритмов дано общее решение проблемы анализа – показано, что их графики есть регулярными функциями относительно одного довольно естественного базиса.

Литература

1 Зубенко В.В. Про темпоральні процедури та алгоритми / В.В. Зубенко // Вісн. КНУ ім. Тараса Шевченка. Серія: Кібернетика. – 2005. – № 6. – С. 20-26.
2 Успенский В.А. Теория алгоритмов: основные открытия и приложения / В.А. Успенский, А.Л. Семенов. – М. : Наука. Гл. ред. физ.-мат. лит., 1987. – 288 с.
3 Зубенко В.В. Періодичні темпоральні алгоритми / В.В. Зубенко // Вісн. Київ. ун-ту. Серія: Кібернетика. – 2006. – № 7. – С. 19-23.
4 Зубенко В.В. Про регуляризацію монотонних табличних алгоритмів / В.В. Зубенко // Штучний інтелект. – 2006. – № 3.

В.В. Зубенко

До поняття функції як обчислювальної процедури

Робота присвячена формальному уточненню загального поняття функції як обчислювальної процедури. При цьому вибирається гранично можливий рівень абстракції, який умовно можна було б назвати пропозиційним: усі об’єкти трактуються виключно як теоретико-множинні «чорні скриньки». Поняття функції-процедури дозволяє уточнити і загальне поняття абстрактного алгоритму – воно стає його похідним. Розглядаються загальні властивості таких алгоритмів. Показано, що функції-процедури і абстрактні алгоритми замкнені відносно регулярних композицій. Для табличних алгоритмів подано загальний розв’язок проблеми аналізу.

V.V. Zubenko

To the Concept of Function as a Computational Procedure

The research is dedicated to the formal refinements of general notion of function as a computational procedure. The strongest level of abstraction is adopted: all objects are treated as “black box”. The notion of function as a procedure makes it possible to revise generic notion of abstract algorithm, as it becomes it’s derivative. The general properties of such algorithms are considered. It is proven that classes of function-procedures and abstract algorithms are closed under regular compositions. A generic solution for the tabulated algorithms analysis problem is proposed.

Статья поступила в редакцию 19.07.2010.